SAVER: SNARK-compatible Verifiable Encryption

Jiwon Lee[1], Jaekyoung Choi[2], Jihye Kim[3], and Hyunok Oh[4]

[1] Samsung Research, Seoul, Korea, jiwonsec.lee@samsung.com
[2] Zkrypto Inc., Seoul, Korea, cjk@zkrypto.com
[3] Kookmin University, Seoul, Korea, jihyek@kookmin.ac.kr
[4] Hanyang University, Seiou, Korea, hoh@hanyang.ac.kr

Abstract. In applications involving zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK), there often exists a requirement for the proof system to be combined with encryption. As a typical example, a user may want to encrypt his identity, while proving that his identity satisfies a given authorized function (e.g. credit checks). However, depending on the functionalities and message types, including encryption constraints inside the SNARK input may lead to impractically large proving time and CRS sizes.
In this paper, we propose a *SNARK-compatible verifiable encryption* or in short *SAVER*, which is a novel encrypt-and-prove approach to modularize the encryption apart from SNARK circuits. The SAVER holds many useful properties. It is *SNARK-compatible*: the encryption scheme is combined with an existing SNARK, in a way that the encryptor can prove pre-defined properties while encrypting the message apart from SNARKs. It is *additively-homomorphic*: the ciphertext holds a homomorphic property by following an ElGamal-like design. It is a *verifiable encryption*: one can verify arbitrary properties of encrypted messages by using the combined SNARK. It provides a *verifiable decryption*: the public can verify that the plaintext claimed by decryptor is equal to the original decryption of ciphertext. It also provides *rerandomization*: the proof and the ciphertext can be rerandomized as independent objects so that even the encryptor (or prover) herself cannot identify the origin.

Keywords: zk-SNARK, verifiable encryption, encrypt-and-prove

1 Introduction

Verifiable encryption (VE) [3, 1] is a cryptographic system where the encrypted data provides a proof that can guarantee publicly-defined properties. It can be a useful primitive in trust-based protocols, such as group signatures or key escrow services. The verifiable property varies depending on the nature of the application. For instance, in the group signature, the verifiable encryption is used for the signer to encrypt and prove its identity commitment, which is evidence for detecting the malicious signer in case of treachery. In the key escrow systems

where users deposit their keys to the trusted party, the verifiable encryption can let users prove their legitimacy of encrypted keys to the others.

The zero-knowledge proof (ZKP) system is a primitive where one can prove a knowledge for some pre-defined relation $\mathcal{R}$, without revealing any other information. As in previous definitions [3], the verifiable encryption can be also viewed as an encryption scheme combined with the ZKP system, by considering the encrypted message as an instance which satisfies the pre-defined relation $\mathcal{R}$. But the early version only focused on verifying the *validity* of the message, i.e., the ciphertext is generated from a correct message, which limits the practicality in various applications.

***Generic* VE from zk-SNARK.** The zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) [19, 10, 12, 2, 14, 17], modern type of succinct ZKPs, allow proving more general relations apart from the ciphertext generation. If we consider a zk-SNARK with general relation inputs, it is possible to construct *generic* verifiable encryption, which can prove *any* desired properties of the message. For instance, a user may encrypt his identity, while proving that his identity satisfies some authorized functions (e.g. credit checks).

Unfortunately, a naive combination of zk-SNARK and encryption suffer from efficiency issues, especially when we consider a functional encryption which may involve heavy cryptographic operations. When encrypting the message while proving it, one should guarantee that she uses the same message for encryption and proof. To ensure this message consistency, the whole encryption process must be included in the zk-SNARK relation, which incurs heavy circuits and large overheads. This problem is well-addressed in *commit-and-prove* system from LegoSNARK [4] framework, which let the user commit for the value ahead of time, and let the pre-published commitment be connected to the zk-SNARK proof gadgets. By using similar approach, we may separate the encryption from the zk-SNARK circuit, and let them be composable when verifying the proof for SNARK and ciphertext. However, LegoSNARK only provides a composability for a commit-carrying system - that is, we do not have any *encrypt-and-prove* scheme that can work as a proof gadget for the LegoSNARK framework. Therefore, we require a new *commit-carrying encryption*, which can work as a modular system for the LegoSNARK framework, similar to the commit-carrying SNARK gadgets.

Generic SAVER. We propose a novel *SAVER: SNARK-compatible verifiable encryption*, a commit-carrying encryption which can be connected to zk-SNARK in a modular way. SAVER is an efficient encrypt-with-prove scheme which supports generic verifiable encryption without including encryption in the circuit. At the same time, SAVER can also be used as a composable encrypt-and-prove (or LegoEncryption) scheme by providing a commitment compatible with the commit-and-prove framework of LegoSNARK [4]. By holding both properties, SAVER can either be used to encrypt while proving some properties of the message (generic verifiable encryption), or to encrypt ahead and prove something about the message later (composability).

The proposed SAVER supports more useful features additional to the basic encryption, listed as follows:
Additive-homomorphism: SAVER is a variation of ElGamal encryption [6], which is additively-homomorphic, i.e., $G^{m_1+m_2} = G^{m_1} \cdot G^{m_2}$. *Verifiable decryption:* a verifiable decryption [3] is a primitive which can convince the verifier that the decrypted message is indeed from the corresponding ciphertext. Likewise, the decryption in SAVER entails a decryption proof, which is verified with a message and a ciphertext to guarantee the validity. This allows the decryptor to prove the correctness of decrypted messages without revealing her secret key. *Rerandomizable encryption:* a rerandomizable encryption [20] is a public-key encryption scheme where the ciphertext can be rerandomized, which can be viewed as a newly-encrypted ciphertext. Likewise, a ciphertext in SAVER can be rerandomized as a new unlinkable ciphertext. Since SAVER outputs a proof as verifiable encryption, the proof is also rerandomized along with the ciphertext.
Applications. SAVER can act as a useful facilitator in many decentralized situations, especially when the encryption results are shared publicly. Apart from the designated/destined receiver who is capable of decrypting the message, more public observers who cannot decrypt the message may at least want to assure that the uploaded ciphertext meets certain regulations (e.g. memberships, formats, relations).
Anonymous Credentials. Suppose that a service provider allows users access after they prove the (zero-knowledge) set membership, but the third-party regulator (e.g. government) wants full traceability of the identity. A user may prove the set membership of identity number to the service provider, while encrypting its raw identity number to the regulator so that the regulator can decrypt and audit when required. *Voting Systems.* In a decentralized voting system, each voter must encrypt its vote for privacy, while proving the universal suffrage (i.e. voting rights and equality). Each vote can be publicly verified, without revealing any provenance or message of the vote. *Digital Contracts with Privacy.* Assuming an official data sharing platform (e.g. government website), two or more parties can commit their encrypted contract online while proving that the contract meets certain government regulations (e.g. deposit amount does not exceed law-enforced limit).
Organization. The rest of the paper proceeds as follows. Section 2 organizes related works. In section 3, we describe some necessary preliminaries and formal definitions. Section 4 presents insights and the formal construction of SAVER, and section 5 shows experiment results of SAVER. In section 6, we draw a conclusion.

2 Related Work

The formal generalized notion of verifiable encryption was introduced by Camenisch and Shoup [3], which concerns the problem of proving properties about encrypted data. They demonstrated many meaningful applications can be achieved by verifiable encryption, with the existence of efficient proof system for proving

the property of the message. Recent succinct proof systems that use pairings for the verification, known as zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) [10, 14, 18, 5], enabled more feasible deployment of verifiable encryption by allowing prover to compile encryption (which is more optimized for SNARK circuits in recent works [15, 16]) combined with the original relation. However, this often led to an unaffordable circuit size, since it increased linear to the relation mingled with non-trivial encryption arithmetics. To mitigate this problem, LegoSNARK [4] introduced a SNARK-composable approach that leads each proof gadgets to prove its own relation, while allowing the gadgets to be verified together in the composable framework. By using an encryption gadget with other proof gadget, one can prove encryption and other relation separately, while composing them in the LegoSNARK verification.

The design of verifiable encryption scheme may vary depending on whether comprehending the relation (i.e. what to prove) as a special-case or general. Applications that treat verifiable encryption as a building block often require nothing more than a simple validity of the message (e.g. range, decryptability). In this context, variety of works propose verifiable encryption with a specific relation, such as verifiable secret sharing [11, 9, 13] which requires range-asserting encryption from unknown users. They focus on verifiable encryption with specific range proofs, which is more efficient, but not applicable to general relation (i.e. not capable of proving properties other than *valid range*). On the other hand, some applications require mutable relation for generality. In this case, they need a general version of verifiable encryption such as LegoSNARK [4] encryption gadget, including the work in this paper.

Table 1: Summary of approaches for achieving generic verifiable encryption, i.e., proving $f(M)$ and encrypting $Enc(M)$ for the same message M.

	All-in-one prove	**LegoSNARK gadgets**	**SAVER**
Composability	*no*	*yes*	*yes*
Relations	$\mathcal{RG}(Enc(M) \mid f(M))$	$\mathcal{RG}(Enc(M)) \mid \mathcal{RG}(f(M))$	$\mathcal{RG}(f(M))$
Total Prove & Enc Time	$\mathsf{Prove}(Enc(M) \mid f(M))$ $+\mathsf{Enc}(M)$	$\mathsf{Prove}(Enc(M)) + \mathsf{Prove}(f(M))$ $+\mathsf{Enc}(M)$	$\mathsf{Prove}(f(M))$ $+\mathsf{Enc}(M)$
CRS Size	$\mathsf{CRS}(Enc) + \mathsf{CRS}(f)$	$\mathsf{CRS}(Enc) + \mathsf{CRS}(f)$	$\mathsf{CRS}(f)$
CT Size	$O(M)$	$O(M)$	$O(M)^{\dagger}$
PK Size	$O(1)$	$O(1)$	$O(M)$

† $\mathcal{CT} = \mathbb{G} \cdot n$, *where* $\mathbb{G}$ = *group element size and* n = *message chunks*
∗ $\mathcal{RG}$ = *relation generator,* f = *arbitrary relation*
∗ Prove = *Prove time,* Enc = *Enc time,* CRS = *CRS size*

Table 1 shows a general comparison of *generic* verifiable encryption techniques, which focus on applications with non-restrictive relation (including encryption). Among the existing works, there exists only two approaches to achieve the generic verifiable encryption - by proving everything (including encryption)

within a single relation, or proving each (encryption and the rest) with a proof gadget and composing them with LegoSNARK [4] - we simply refer to the former approach as *All-in-one prove*, and the latter approach as *LegoSNARK gadgets*.

The only difference between All-in-one prove and LegoSNARK gadgets is that All-in-one prove combines encryption (Enc) and arbitrary relation (f) as a single relation (i.e. not composable), while LegoSNARK separates them as independent proof gadgets (i.e. composable). In both approaches, the size of ciphertext and public key depends on the encryption scheme; assuming standard public key encryption (e.g. RSA-2048), the public key is a fixed constant while the ciphertext size grows linear to the message size.

Compared with All-in-one prove and LegoSNARK gadgets, ciphertext in SAER also grows linear to the message size. More precisely, the ciphertext of SAVER consists of n group elements, where n is the number of message chunks (e.g. 8-bit). For example, when a 2048-bit message is split into 8-bit chunks, SAVER requires $n = 256$ group elements as ciphertexts. Although SAVER has a relatively larger public key compared to the existing standards, it can show much more benefits in total encryption time (including prove) and CRS size, since SAVER does not require proving of the encryption-related constraints.

3 Preliminaries

3.1 Notations

In this section, we define some essential notations. For the simple legibility, we define the term $\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}$ in [10] as $y_i(x)$. Then, we denote $G^{y_i(x)}$ as G_i. We use $\boldsymbol{x}$ or $\{x_i\}$ for the list of elements, which is equivalent to a vector. We also define $[\![X]\!] = span\{X\}$ as a linear combination of $x \in X$, i.e., $[\![X]\!] = \{\sum_{x_i \in X} \eta_i x_i\}$. For any set $[\![X]\!]$, we define $[\![A]\!] \times [\![B]\!] = \{a \cdot b \mid a \in [\![A]\!],\ b \in [\![B]\!]\}$ and $[\![A]\!]^{-1} = \{a^{-1} \mid a \in [\![A]\!]\}$. For any given vectors, $\circ$ represents a Hadamard product (i.e. let $\boldsymbol{a} = (a_1, a_2)$ and $\boldsymbol{b} = (b_1, b_2)$, then $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 \cdot b_1, a_2 \cdot b_2)$) and $\oslash$ represents a Hadamard division($\boldsymbol{a} \oslash \boldsymbol{b} = (a_1/b_1, a_2/b_2)$).

3.2 Relations

Given a security parameter 1^λ, a relation generator $\mathcal{RG}$ returns a polynomial time decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$. For $(\Phi, w) \in \mathcal{R}$ we say w is a witness to the statement (I/O) Φ being in the relation. The statement Φ in SAVER consists of $\Phi = M \cup \hat{\Phi}$ for message statements $\{m_1, \ldots, m_n\}$ by splitting $M = (m_1 || \cdots || m_n)$ and arbitrary statements $\hat{\Phi} = \{\phi_{n+1}, \cdots, \phi_l\}$, where l is the number of statements.

3.3 Bilinear Groups

Definition 1. *A bilinear group generator $\mathcal{BG}$ takes a security parameter as input in unary and returns a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$ consisting of cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of prime order p and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ possibly together with some auxiliary information (aux) such that:*

- *there are efficient algorithms for computing group operations, evaluating the bilinear map, deciding membership of the groups, and for sampling the generators of the groups;*
- *the map is bilinear, i.e., for all* $G \in \mathbb{G}_1$ *and* $H \in \mathbb{G}_2$ *and for all* $a, b \in \mathbb{Z}$ *we have*

$$e(G^a, H^b) = e(G, H)^{ab};$$

- *and the map is non-degenerate (i.e., if* $e(G, H) = 1$ *then* $G = 1$ *or* $H = 1$*).*

3.4 Cryptographic Assumptions

We use *Power Knowledge of Exponent (d-PKE) with Batch Knowledge Check* assumption [7]. In [7] (lemma 2.3), it is proven that the *d-PKE* can be used to *batch* knowledge checks, stated as below:

Assumption 1. $\mathsf{batch-PKE}$ *(Lemma 2.3 of [7]): Assuming the d-PKE the following holds. Fix* $k = poly(\lambda)$*, a constant* t *and an efficiently computable degree* d *rational map* $S : \mathbb{F}^{t+1} \to \mathbb{F}^M$*. Fix any* $i \in [k]$*. For any efficient* $\mathcal{A}$ *there exists an efficient* $\chi_{\mathcal{A}}$ *such that the following holds. Consider the following experiment.* $\alpha_1, \ldots, \alpha_k, \tau \in \mathbb{F}$ *and* $\boldsymbol{x} \in \mathbb{F}^t$ *are chosen uniformly.* $\mathcal{A}$ *is given as input* $[S(\tau, \boldsymbol{x})]$ *and* $\{[\alpha_j \cdot \tau^l]\}_{j \in [k], l \in [0..d]}$ *and outputs a sequence of elements* $([a_1], \ldots, [a_k], [b])$ *in* $\mathbb{G}$*.* $\chi_{\mathcal{A}}$*, given the same input as* $\mathcal{A}$ *together with the randomness of* $\mathcal{A}$ *and* $\{\alpha_j\}_{j \in [k] \setminus \{i\}}$*, outputs* $A(X) \in \mathbb{F}[X]$ *of degree at most* d *such that the probability that both*

1. $\mathcal{A}$ *"succeeded", i.e.,* $b = \sum_{j=1}^{k} \alpha_j \cdot a_j$*. But,*
2. $\chi_{\mathcal{A}}$ *"failed", i.e.,* $a_i \neq A(\tau)$*.*

is $\mathbf{Adv}^{\text{batch-PKE}}_{\mathcal{R}, \mathcal{A}, \chi_{\mathcal{A}}}(\lambda) = negl(\lambda)$*.*

We also introduce a $D-Poly$ assumption, which is a decisional version of the computational Poly assumption originated from [12], formally defined as below:

Assumption 2. $\mathsf{D-Poly}$*: Let* $\mathcal{A}$ *be a PPT adversary, and define the advantage* $\mathbf{Adv}^{D-Poly}_{\mathcal{BG}, d(\lambda), q(\lambda), \mathcal{A}}(\lambda) = Pr[\mathcal{G}^{D-Poly}_{\mathcal{BG}, d(\lambda), q(\lambda), \mathcal{A}}] - \frac{1}{2}$ *where* $\mathcal{G}^{D-Poly}_{\mathcal{BG}, d(\lambda), q(\lambda), \mathcal{A}}$ *is defined as below and* Q_1, Q_2 *is the set of polynomials* $g_i(X_1, \ldots, X_q), h_i(X_1, \ldots, X_q)$ *queried to* $\mathcal{O}^1_{G, \boldsymbol{x}}, \mathcal{O}^2_{H, \boldsymbol{x}}$*.*

$\underline{\mathsf{MAIN}\ \mathcal{G}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)}$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux) \leftarrow \mathcal{BG}(1^\lambda);$

$G \leftarrow \mathbb{G}_1; H \leftarrow \mathbb{G}_2; \boldsymbol{x} \leftarrow (\mathbb{Z}^*_p)^q$

$g_c(X_1, \ldots, X_q) \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$

$where\ g_c(\boldsymbol{x}) \notin [\![Q_1]\!] \times [\![Q_2]\!] \times [\![Q_2]\!]^{-1}$

$set\ T_1 \leftarrow G^{g_c(\boldsymbol{x})}, T_0 \xleftarrow{\$} \mathbb{G}_1$

$b \leftarrow \{0,1\}, T = T_b$

$b' \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(T)$

$return\ 1\ if\ b = b'$

$else\ return\ 0$

$\underline{\mathcal{O}^1_{G,\boldsymbol{x}}(g_i)}$

$assert\ g_i \in \mathbb{Z}^*_p[X_1, \ldots, X_q]$

$assert\ deg(g_i) \leq d$

$return\ G^{g_i(\boldsymbol{x})}$

$\underline{\mathcal{O}^2_{H,\boldsymbol{x}}(h_j)}$

$assert\ h_j \in \mathbb{Z}^*_p[X_1, \ldots, X_q]$

$assert\ deg(h_j) \leq d$

$return\ H^{h_j(\boldsymbol{x})}$

The $(d(\lambda), q(\lambda)) - D - Poly$ assumption holds relative to $\mathcal{BG}$ if for all PPT adversaries $\mathcal{A}$, we have $\mathbf{Adv}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)$ is negligible in λ.

In the D-Poly game, the adversary acts similarly as in computational Poly game, except that it queries a challenge polynomial and guesses the nature of the output (i.e. whether the output is generated from the polynomial or from an independent random). In this case, the restriction for the challenge $g_c \notin [\![Q_1]\!]$ is not sufficient where $Q_1 = \{g_1, \ldots, g_I\}$. For example, the adversary should not have $H^{g_c(\boldsymbol{x})}$; otherwise it can check whether the received challenge T is $G^{g_c(\boldsymbol{x})}$ or a random group element by applying pairings (i.e. check the nature of T by $e(T, H^{g_c(\boldsymbol{x})})$). This problem is similar to the decisional BDH assumption: it cannot follow the standard DDH as $(g^a, g^b, T_0 \leftarrow g^z, T_1 \leftarrow g^{ab}, b \leftarrow \{0,1\} \quad | \quad b' \leftarrow \mathcal{A}(g^a, g^b, T))$, because the adversary can test if $e(g^a, g^b) \stackrel{?}{=} e(g, T)$. Thus, the restriction should be extended to $H \in \mathbb{G}_2$, to prevent the adversary from obtaining the span of $g_c(\boldsymbol{x})$ in $\mathbb{G}_2$.

3.5 Definition of SAVER

We represent the combined definition of our SAVER: SNARK-compatible verifiable encryption - which captures the properties of verifiable encryption Π_{VE} [3] and zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) Π_{snark} [10] at the same time. As a generic verifiable encryption, SAVER accepts any relation representable in arithmetic rank-1 constraint system (R1CS). The intuitive goal of SAVER is that Enc for a message M must satisfy IND-CPA of standard public key encryption, while $\mathsf{Verify}_{\mathsf{E}}\mathsf{nc}$ of proof π and ciphertext $\mathcal{CT}$ (as proof I/O) must satisfy the SNARK-like knowledge soundness, i.e., the message M hidden in the ciphertext $\mathcal{CT}$ is within the relation. In addition, SAVER also defines other useful properties: verifiable decryption Π_{VD} (section A.4) and rerandomizable encryption Π_{RR} (section A.5).

Definition 2. *For any arbitrary zk-SNARK relation $\mathcal{R}$ (also noted as* relation*), the* SAVER *consists of seven polynomial-time algorithms as follows:*

- $CRS \leftarrow \mathsf{Setup}(\mathsf{relation})$: *takes an arbitrary relation $\mathcal{R}$ as an input, and outputs the corresponding common reference string CRS.*
- $SK, PK, VK \leftarrow \mathsf{KeyGen}(CRS)$: *takes a CRS as an input, and outputs the corresponding secret key SK, public key PK, verification key VK.*
- $\pi, \mathcal{CT} \leftarrow \mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$: *takes CRS, a public key PK, a message $M = m_1, \ldots, m_n$, a zk-SNARK statement $\hat{\Phi} = \{\phi_{n+1}, \ldots, \phi_l\}$, and a witness w as inputs, and outputs a proof π and a ciphertext $\mathcal{CT} = (c_0, \cdots, c_n, \psi)$.*
- $\pi', \mathcal{CT}' \leftarrow \mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$: *takes a public key PK, a proof π, a ciphertext $\mathcal{CT}$ as inputs, and outputs a new proof π' and a new ciphertext $\mathcal{CT}'$ with fresh randomness.*
- $0/1 \leftarrow \mathsf{Verify\_Enc}(CRS, PK, \pi, \mathcal{CT}, \hat{\Phi})$: *takes CRS, a public key PK, a proof π, a ciphertext $\mathcal{CT}$, and a statement $\hat{\Phi} = \{\phi_{n+1}, \ldots, \phi_l\}$ as inputs, and outputs 1 if $\mathcal{CT}, \hat{\Phi}$ is in the relation $\mathcal{R}$, or 0 otherwise.*
- $M, \nu \leftarrow \mathsf{Dec}(CRS, SK, VK, \mathcal{CT})$: *takes CRS, a secret key SK, a verification key VK, and a ciphertext $\mathcal{CT} = (c_0, \cdots, c_n, \psi)$ as inputs, and outputs a plaintext $M = m_1, \ldots, m_n$ and a decryption proof ν.*
- $0/1 \leftarrow \mathsf{Verify\_Dec}(CRS, VK, M, \nu, \mathcal{CT})$: *takes CRS, a verification key VK, a message M, a decryption proof ν, and a ciphertext $\mathcal{CT}$ as inputs, and outputs 1 if M is a valid decryption of $\mathcal{CT}$, or 0 otherwise.*

It satisfies completeness, indistinguishability, encryption knowledge soundness, rerandomizability, decryption soundness, perfect zero-knowledge as below:

Completeness: The completeness of SAVER must satisfy the completeness of Π_{snark}, Π_{VE}, Π_{VD} and Π_{RR} altogether. For all $\lambda \in \mathbb{N}$, $\mathcal{R}$, $(\Phi, w) \in \mathcal{R}$, proof π, ciphertext $\mathcal{CT}$, message M, decryption proof ν, the completeness must satisfy that:

$$Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), \pi \leftarrow \mathsf{Prove}(CRS, \Phi, w) : \mathsf{Vfy}(CRS, \Phi, \pi) = 1] = 1$$
$$Pr[(\pi, \mathcal{CT}) \leftarrow \mathsf{Enc}(PK, M), M \in \mathcal{R} : \mathsf{Verify\_Enc}(VK, \pi, \mathcal{CT}) = 1] = 1$$
$$Pr[(M, \nu) \leftarrow \mathsf{Dec}(SK, \mathcal{CT}), \mathcal{CT} = \mathsf{Enc}(PK, M) : \mathsf{Verify\_Dec}(VK, M, \nu, \mathcal{CT}) = 1] = 1$$
$$Pr[\mathcal{CT} = \mathsf{Enc}(PK, M), \mathsf{Rerandomize}(PK, \mathcal{CT}) : \mathsf{Dec}(SK, \mathcal{CT}') = \mathsf{Dec}(SK, \mathcal{CT})] = 1$$

Indistinguishability: The indistinguishability is also known as semantic security (IND-CPA). The IND-CPA of the SAVER must satisfy that of Π_{VE} and Π_{VD}, which is defined by an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ via following game.

Setup: The challenger $\mathcal{C}$ runs $\mathsf{Setup}(\mathsf{relation})$ to obtain CRS, τ, and share CRS, τ and statements $\hat{\Phi}$ to $\mathcal{A}$. Note that the adversary $\mathcal{A}$ is given the trapdoor $\tau = \{\alpha, \beta, \gamma, \delta\}$ as an additional information, since ability to simulate the proof does not affect the security of the ciphertext indistinguishability.

KeyGen: $\mathcal{C}$ runs $\mathsf{KeyGen}(CRS)$ to obtain a secret key SK, a public key PK, and a verification key VK. Then, $\mathcal{C}$ gives PK,VK to $\mathcal{A}$.

$\mathcal{O}_\nu$ phase 1: If the message is decrypted, the decryption proof ν is also revealed. Therefore, $\mathcal{A}$ may request decryption proof for M as an additional information since knowing M may indicate it is already decrypted. For the polynomial-time, $\mathcal{A}$ may issue decryption proof query as M_i, to obtain the corresponding ciphertext $\mathcal{CT}_i$ and a decryption proof ν_i. $\mathcal{C}$ generates $\pi_i, \mathcal{CT}_i$ by running $\mathsf{Enc}(CRS, PK, M_i, \hat{\Phi}; w)$, generates ν_i by running $\mathsf{Dec}(CRS, SK, VK, \mathcal{CT}_i)$, and returns $(\pi_i, \mathcal{CT}_i, \nu_i)$ to $\mathcal{A}$.

Challenge: For the challenge, $\mathcal{A}$ outputs two messages M_0 and M_1. $\mathcal{C}$ picks $b \in \{0, 1\}$ to choose M_b, generates $\pi, \mathcal{CT}$ by running $\mathsf{Enc}(CRS, PK, M_b, \hat{\Phi}; w)$, and returns $\pi, \mathcal{CT}$ to $\mathcal{A}$.

$\mathcal{O}_\nu$ phase 2: $\mathcal{A}$ can continue to issue encryption queries M_j, same as $\mathcal{O}_\nu$ phase 1. The only restriction is that $M_j \notin \{M_0, M_1\}$.

Guess: $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ for b, and wins the game if $b = b'$.

Let $\mathbf{Adv}^{ind}_{\mathsf{SAVER},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the above game. For a negligible function ϵ, it is IND-CPA secure if for any adversary $\mathcal{A}$ we have that $|\mathbf{Adv}^{ind}_{\mathsf{SAVER},\mathcal{A}}(\lambda) - 1/2| < \epsilon$.

Encryption Knowledge Soundness: The encryption knowledge soundness is a combined definition of computational knowledge soundness in Π_{snark} and encryption soundness in Π_{VE}. It is formally defined as follows:

$$\begin{aligned}\mathbf{Adv}^{sound}_{\mathsf{SAVER},\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), (PK, SK, VK) \leftarrow \mathsf{KeyGen}(CRS),\\ (\pi^*, \mathcal{CT}^*, \hat{\Phi}^*) \leftarrow \mathcal{A}(CRS, PK, VK), (M, w) \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) :\\ \mathsf{Verify\_Enc}(CRS, \pi^*, \mathcal{CT}^*, \hat{\Phi}^*) = 1 \wedge (\mathsf{Dec}(\mathcal{CT}^*) \neq M \vee (M, \hat{\Phi}^*, w) \notin \mathcal{R})] = negl(\lambda).\end{aligned}$$

Rerandomizability: The rerandomizability is extended from Π_{RR}, to include π as follows: for all M and $\pi, \mathcal{CT}$ in the support of $\mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$, the distribution of $\mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$ is identical to another round of $\mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$.

Perfect Decryption Soundness: Equivalent to the perfect decryption soundness in Π_{VD}, i.e., formally defined as follows:

$$\begin{aligned}\mathbf{Adv}^{sound}_{\Pi_{\mathsf{VD}},\mathcal{A}}(\lambda) = Pr[(M^*, \nu^*, \mathcal{CT}^*) \leftarrow \mathcal{A}(SK, PK, VK) :\\ \mathsf{Verify\_Dec}(VK, M^*, \nu^*, \mathcal{CT}^*) = 1 \wedge \mathsf{Dec}(SK, \mathcal{CT}^*) \neq M^*] = 0.\end{aligned}$$

Perfect Zero-Knowledge: Equivalent to the perfect zero-knowledge in Π_{snark}, i.e., there exists a simulator that does not know the witness but has some trapdoor information that enables it to simulate proofs.

4 Proposed SAVER

4.1 Main Idea

Before representing the concrete algorithms, we first introduce an intuitive idea behind the construction. As a starter, we observe the nature of pairing-based non-interactive arguments (i.e. SNARKs), which is well-described in Groth16 paper [10]. An analogy from Groth16 analysis is that SNARKs (including the QAP-based Groth16 construction itself) can obtain succinct proof by balancing statements (i.e. I/O) and witnesses in the pairing-based equality check of $A \cdot B = C$.

$$A = \alpha + \Sigma_{i=0}^{m} a_i u_i(x) + r\delta \qquad B = \beta + \Sigma_{i=0}^{m} a_i v_i(x) + s\delta$$

$$C = \frac{\Sigma_{i=l+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + As + Br - rs\delta$$

The equation above represents the essential elements of Groth16 construction, where statements are $a_1, \cdots, a_l \in \mathbb{F}^l$ and witnesses are $a_{l+1}, \cdots, a_m \in \mathbb{F}^{m-l}$. To balance the equality check of $A \cdot B = C$, both statements and witnesses $a_1, \cdots, a_m$ are batched into the coefficients of polynomials in A and B, while only witnesses $a_{l+1}, \cdots, a_m$ are batched into the coefficients of polynomials in C. Then in verification, statements $a_1, \cdots, a_l$ are *appropriately (as span of $y_i(x)$)*[5] combined into C to reassemble all spans of $a_1, \cdots, a_m$, so that it can cancel out all the $a_1, \cdots, a_m$ spans combined in A and B.

Message Statement transformable to ElGamal Ciphertext. Orthogonal to encryption, when we consider proving some properties of the message m_i, the message m_i itself becomes a statement. In verification, these message statements m_i are combined into the element of C as a span of $y_i(x)$, i.e., $a_i \cdot y_i(x)$. The concrete verification algorithm of Groth16 for the message statements m_i is represented as below:

$$e(A, B) = e(G^{\alpha}, H^{\beta}) \cdot e(\prod_{i=0}^{l} G_i^{m_i}, H^{\gamma}) \cdot e(C, H^{\delta})$$

Notice that, the message statement lies in the group element as ${G_i}^{m_i}$, which takes a form of ElGamal primitive - if we add a random blinding factor to the element as ${G_i}^{m_i} \cdot {X_i}^{r}$, it becomes a short discrete-log ElGamal ciphertext for small message blocks m_i (i.e. message is chunked). When the blinding factor is eliminated (by some secrets), the decryptor can retrieve the message by solving a short discrete-log (e.g. 8-bits) from ${G_i}^{m}$. So if we maintain the equality check while allowing these *ciphertext-transformed message statements* to work as an original input to the verifying statements, the verification itself implies that the message behind the ciphertext satisfies the original soundness for arguments. In order to make this actually happen though, we need to find a graceful way of generating and neutralizing these blinding factors ${X_i}^{r}$ which distorts the original equality check (while assuring the semantic security). SAVER adds a $G^{-\gamma}$ in the

[5] We simplify $\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}$ as $y_i(x)$, as mentioned in section 3.1.

CRS as a neutralizing factor for $y_i(x) = \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}$, to blend it in the public keys in a way that satisfy both indistinguishability of the ciphertext and neutralization of the equality check - which is the main contribution of SAVER.

Replaceability of the underlying construction. In SAVER, we design the concrete verifiable encryption scheme based on the Groth16 [10] construction. However, our intuitive idea shows that any linear proof systems (including polynomial commitment variants like Sonic [18] or Plonk [8]) that share the idea of the *equality-check* - which combine input/output statements in some structured reference strings of group elements (i.e. ${G_i}^{a_i}$) and plugged into the verification - can benefit from the idea of extending the statement elements ${G_i}^{a_i}$ to a series of ElGamal ciphertexts ${G_i}^{m_i} \cdot {X_i}^{r}$. Nevertheless, designing another encryption based on a different verification scheme while maintaining provable security is a non-trivial work; exploring the applicability of the SAVER intuition and analyzing whether it is possible to come up with a general extension technique (i.e. extending any linear proofs to verifiable encryption) would be an interesting future work.

4.2 SAVER Construction

We now represent a formal construction of the proposed SAVER. In SAVER, a message M is split into n blocks as $M = (m_1 || \cdots || m_n)$, to form a vector $M = \{m_1, \ldots, m_n\}$[6]. A ciphertext $\mathcal{CT}$ consists of $n+2$ blocks as $\mathcal{CT} = \{c_0, \cdots, c_n, \psi\}$, where c_0 contains the random, ψ contains a commitment, and the remaining c_i contains an encryption of m_i for $1 \leq i \leq n$. Within the construction, we work with $\{m_1, \ldots, m_n\}$, assuming that M is already parsed to $M = (m_1 || \cdots || m_n)$.

Algorithm 1 represents the formal construction of SAVER. The term relation denotes an arbitrary relation $\mathcal{R}$ for the zk-SNARK, and the terms of α, β, γ, and δ within the functions come from CRS (common reference string) of the adopted zk-SNARK scheme [10]. In case of encrypt-and-prove, there is a possibility that the relation has not been determined yet (when encrypting only ahead of time); in this case, the relation can be assumed as an empty circuit (i.e. $G^{\delta}, G_i, G^{\gamma} \xleftarrow{\$} \mathbb{G}_1$).

SAVER receives any relation which consists of two I/O statements. Statements $m_1, \ldots, m_n$ will be encrypted while statements $\phi_{n+1}, \ldots, \phi_l$ will be used as normal I/O statements in plaintext. For the given relation, Setup generates CRS using the adopted zk-SNARKs scheme, with additional $G^{-\gamma}$. KeyGen generates a private key, a public key, and a verification key. Enc encrypts messages $m_1, \ldots, m_n$ and generates a proof π of statement $\Phi = (m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l)$. To check the truth of statement Φ, Verify_Enc takes π and $\mathcal{CT}$ as inputs for verification. Rerandomize does rerandomization of the given ciphertext and the proof. Note that the rerandomized proof is a valid proof of the statement. Dec decrypts the ciphertext $\mathcal{CT}$ by performing decryption for each block $c_1, \ldots, c_n$, to output $m_1, \ldots, m_n$ and a decryption proof ν. The original message M can be restored

[6] These are pre-defined relation (section 3.2) with a fixed messsage chunk size (e.g. 8-bits), which does not require an addtional range proof.

Algorithm 1 SAVER construction

$\mathsf{relation}(m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$:

$\ldots$

$\mathsf{Setup}(\mathsf{relation})$:

$C\hat{R}S \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathsf{relation})$
$CRS \leftarrow C\hat{R}S \cup \{G^{-\gamma}\}$
return CRS

$\mathsf{KeyGen}(CRS)$:

$\{s_i\}_{i=1}^n, \{v_i\}_{i=1}^n, \{t_i\}_{i=0}^n, \rho \xleftarrow{\$} \mathbb{Z}_p^*$
$PK \leftarrow (G^{\delta}, \{G^{\delta s_i}\}_{i=1}^n, \{G_i^{t_i}\}_{i=1}^n, \{H^{t_i}\}_{i=0}^n, G^{\delta t_0} \prod_{j=1}^n G^{\delta t_j s_j}, G^{-\gamma \cdot (1+\sum_{j=1}^n s_j)})$
$SK \leftarrow \rho$
$VK \leftarrow (H^{\rho}, \{H^{s_i v_i}\}_{i=1}^n, \{H^{\rho v_i}\}_{i=1}^n)$
return (SK, PK, VK)

$\mathsf{Enc}(CRS, PK, m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$:

let $PK = (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathcal{CT} = (X_0^r, X_1^r G_1^{m_1}, \ldots, X_n^r G_n^{m_n}, \psi = P_1^r \cdot \prod_{j=1}^n Y_j^{m_j})$
$\hat{\pi} = (A, B, C) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(CRS, m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$
$\pi \leftarrow (A, B, C \cdot P_2^r)$
return $(\pi, \mathcal{CT})$

$\mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$:

parse $\pi = (A, B, C)$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
let $PK = (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
$r', z_1, z_2 \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathcal{CT}' \leftarrow (c_0 \cdot X_0^{r'}, \ldots, c_n \cdot X_n^{r'}, \psi \cdot P_1^{r'})$
$\pi' \leftarrow (A^{z_1}, B^{z_1^{-1}} \cdot H^{\delta \cdot z_2}, C \cdot A^{z_1 z_2} \cdot P_2^{r'})$
return $(\pi', \mathcal{CT}')$

$\mathsf{Verify\_Enc}(CRS, PK, \pi, \mathcal{CT}, \phi_{n+1}, \cdots, \phi_l)$:

parse $\pi = (A, B, C)$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
let $PK = (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
assert $\prod_{i=0}^n e(c_i, Z_i) = e(\psi, H)$
assert $e(A, B) = e(G^{\alpha}, H^{\beta}) \cdot e(\prod_{i=0}^n c_i \cdot \prod_{i=n+1}^l G_i^{\phi_i}, H^{\gamma}) \cdot e(C, H^{\delta})$

as $M = (m_1 || \ldots || m_n)$. The honest decryption of $\mathcal{CT}$ can be proved by calling $\mathsf{Verify\_Dec}$ with a message M and a decryption proof ν.

The ciphertext $\mathcal{CT}$ in SAVER satisfies additive-homomorphic property, for each individual message block. Given $\mathcal{CT} = (X_0^r, \{X_i^r G_i^{m_i}\}_{i=1}^n, P_1^r \prod_{j=1}^n Y_j^{m_j})$ and $\mathcal{CT}' = (X_0^{r'}, \{X_i^{r'} G_i^{m'_i}\}_{i=1}^n, P_1^{r'} \prod_{j=1}^n Y_j^{m'_j})$, it is easy to see that $\mathcal{CT} \cdot \mathcal{CT}' = (X_0^{r+r'}, \{X_i^{r+r'} G_i^{m_i+m'_i}\}_{i=1}^n, P_1^{r+r'} \prod_{j=1}^n Y_j^{m_j+m'_j})$, which satisfies additive-homomorphism. Note that the merged message may not be homomorphic, due to the carry-bits.

$\mathsf{Dec}(CRS, SK, VK, \mathcal{CT})$:
parse $SK = \rho$, $VK = (V_0, \{V_i\}_{i=1}^{n}, \{V_i\}_{i=n+1}^{2n})$, and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
for $i = 1$ **do** to n
$\frac{e(c_i, V_{n+i})}{e(c_0, V_i)^{\rho}} = e(G_i, V_{n+i})^{m_i}$
compute a short discrete log of $e(G_i, V_{n+i})^{m_i}$ to obtain m_i
end for
$\nu \leftarrow c_0^{\rho}$
return $(m_1, \ldots m_n, \nu)$

$\mathsf{Verify\_Dec}(CRS, VK, m_1, \ldots m_n, \nu, \mathcal{CT})$:
parse $VK = (V_0, \{V_i\}_{i=1}^{n}, \{V_i\}_{i=n+1}^{2n})$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
assert $e(\nu, H) = e(c_0, V_0)$
for $i = 1$ **do** to n
assert $\frac{e(c_i, V_{n+i})}{e(\nu, V_i)} = e(G_i, V_{n+i})^{m_i}$
end for

4.3 Encrypt-and-Prove

The encrypt-with-prove construction in section 4.2 is also a commit-carrying encryption: it includes an encrypt-and-prove scheme which allows modular composition to other commit-carrying systems. In this section, we show that the commitment ψ in the ciphertext $\mathcal{CT}$ is identical to the Pedersen vector commitment c in LegoSNARK [4]'s commit-and-prove, which implies that the SAVER construction (algorithm 1) is a commit-carrying encryption (appendix A.3). Then we briefly show that $\mathcal{CT}$ and ψ can be the inputs of CP-SNARK $\mathsf{CP}_{\mathsf{link}}$ (appendix A.2).

Commit-carrying encryption. When observing algorithm 1, it is obvious that SAVER is a commit-carrying encryption since the ciphertext $\mathcal{CT}$ already includes a commitment ψ. We show that ψ is identical to the Pedersen vector commitment of the commit-carrying SNARKs. Recall that the Pedersen vector commitment in a commit-carrying SNARK is constructed as $c = h^o \cdot h_1^{u_1} \cdots h_n^{u_n}$ for a random $o \xleftarrow{\$} \mathbb{Z}_p^*$, messages $u_1, \cdots, u_n$, and random generators $h, h_1, \cdots, h_n$. The commitment ψ in SAVER is constructed as $\psi = P_1^r \cdot \prod_{j=1}^{n} Y_j^{m_j}$. Notice that $P_1 = G^{\delta t_0} \prod_{j=1}^{n} G^{\delta t_j s_j}$ can be viewed as a random generator h with respect to the randoms t_i and s_i, and $\{Y_i = G_i^{t_i}\}_{i=1}^{n}$ can be also viewed as random generators $h_1, \cdots, h_n$ with respect to the random t_i. Since the message $m_1, \cdots, m_n$ and random r for the encryption correspond to the message $u_1, \cdots, u_n$ and random o for the commitment, $\psi = P_1^r \cdot \prod_{j=1}^{n} Y_j^{m_j}$ can be considered as Pedersen vector commitment $c = h^o \cdot \prod_{j=1}^{n} h_j^{u_j}$ by matching $h = P_1, o = r, h_j = Y_j, u_j = m_j$.

CP-SNARK composability. We briefly show how to connect algorithm 1 as a commit-carrying system for the CP-SNARK CP. Recall that the CP protocol let the prover prove the relation with the commitment key ck as follows:

$$crs := (ek, vk) \leftarrow \mathsf{KeyGen}(\mathsf{c}k, R)$$
$$\pi \leftarrow \mathsf{Prove}(ek, x, (c_j)_{j\in[l]}, (u_j)_{j\in[l]}, (o_j)_{j\in[l]}, \omega)$$

For the ck, we may assume P_1 and $\{Y_i\}_{i=1}^n$ as the commitment key ck since they are the ingredients for constructing the commitments $\{\psi_j\}_{j\in[l]}$. Also, in Prove and $\mathsf{VerProof}$, notice that we already showed the commitments $\{c_j\}_{j\in[l]}$ and random o are identical to the commitment $\{\psi_j\}_{j\in[l]}$ and random $\{r_j\}_{j\in[l]}$ from SAVER systems (or other cc-SNARK systems). Therefore, it is straightforward to run the CP by matching the inputs as $\{c_j = \psi_j\}, \{o_j = r_j\}, \{\boldsymbol{u_j} = (m_1, \cdots, m_n)\}$ for $j \in [l]$.

Conceptual Benefits. In the viewpoint of commit-and-prove, encrypt-and-prove can provide more general extension of *selective disclosure*. In the encrypt-and-prove, the encrypted ciphertext can also work as a commitment since the commit-carrying encryption has an efficient function $f_c(x)$ that can output commitment c. Therefore, when the prover outputs the commit-carrying ciphertext, it can disclose whole data to the secret key holder, while only disclosing proved statements to the other public the same as commit-and-prove. As a simple example, assume a blockchain contract management system where Alice and Bob encrypt a contract to each other. Alice wants to let Bob (who has a secret key) see the whole data, but she also wants to prove some restricted statements publicly. In this case, Alice can upload the ciphertext as a commitment of commit-and-prove; Bob can decrypt the ciphertext, and Alice can use the ciphertext as a commitment later for proving various statements.

4.4 Security Proof

To satisfy the definition of SAVER, the scheme should satisfy *completeness*, *indistinguishability*, *encryption knowledge soundness*, *rerandomizability*, and *perfect zero-knowledge*. Especially, the *indistinguishability* (IND-CPA) is held by $\mathsf{D-Poly}$ (Assumption 2); intuitively, $\mathsf{D-Poly}$ states that no PPT adversary $\mathcal{A}$ can determine whether given $\mathbb{G}_1$ group element T is generated from $\mathbb{G}_1, \mathbb{G}_2$ span polynomials $G^{g_c(x)}$ or randomly generated from $\mathbb{G}_1$, where SAVER ciphertexts are mapped to the $\mathbb{G}_1, \mathbb{G}_2$ span polynomials $G^{g_c(x)}$. For the *encryption knowledge soundness*, $\mathsf{batch-PKE}$ (Assumption 3) states that no PPT adversary $\mathcal{A}$ could have generated a G^b that passes $G^b = G^{\alpha_0(x)\cdot a_0} \cdots G^{\alpha_n(x)\cdot a_n}$ other than using the span of G^x, which enables the extraction of all coefficients from batch polynomials $\alpha_i(x)$ equivalent to the SAVER message I/O.

Due to the space limit, rest of the formal proofs are provided in the full version.

5 Experiment

We test the efficiency of SAVER by implementing the construction on the Ubuntu 18.04 machine with Intel-i5 (3.4GHz) quad-cores and 24GB memory. Since SAVER is a generic verifiable encryption for any arbitrary relations, the relation can be defined flexibly depending on the user's requirements. We assigned *hashing* as a sample relation, which can be widely extended to other applications such as set of membership proofs. For a given SHA-256 hashed value

$\mathcal{H}$, the defined relation guarantees that $\mathcal{H} = hash(M)$ and $\mathcal{CT} = encrypt(M)$ for the same message M. We set the SAVER message block size as $|m| = 32bits$, and use Groth16 zk-SNARK [10] for the proof system as in section 4.2 based on libsnark [21].

Table 2: SAVER encryption performance for SHA-256 relation (approximately 25,000 constraints, a little dependency on input wires).

time	$\|M\|$ (*bits*)				*size*	$\|M\|$ (*bits*)			
	256	512	1024	2048		256	512	1024	2048
Setup	$2.67s$	$2.67s$	$2.69s$	$2.72s$	CRS	$16MB$	$16MB$	$16MB$	$16MB$
KeyGen	$0.01s$	$0.02s$	$0.04s$	$0.09s$	SK	$32B$			
Enc (*sep*)	$1.6ms$	$2.4ms$	$7.4ms$	$8.8ms$	PK	$1246B$	$2321B$	$4465B$	$8753B$
Π_{snark}.Prove	$0.73s$	$0.73s$	$0.73s$	$0.74s$	VK	$1126B$	$2184B$	$4296B$	$8520B$
Verify_Enc	$8.2ms$	$12.7ms$	$21.7ms$	$39.8ms$	$\mathcal{CT}$	$477B$	$749B$	$1293B$	$2381B$
Dec	$37.7ms$	$75.2ms$	$149.7ms$	$300.4ms$	π	$128B$			
Verify_Dec	$14.8ms$	$28.3ms$	$55.5s$	$110.1ms$	ν	$32B$			
Rerandomize	$0.02ms$	$0.03ms$	$0.04ms$	$0.06ms$					

$*$ $|M|$ = *message size,* $|m| = 32$ *bits,* Π_{snark} =*Gro16 [10]*

Table 2 lists the execution time for each algorithm step, and size for the generated results. To examine the growth rate of encryption costs per message blocks in SAVER, we vary the message size from 256 bits to 2048 bits while fixing the message block size as $|m| = 32bits$ for all message spaces. For example, 256-bit M consists of 8 blocks of messages. The block size determines the ciphertext size and decryption time. A larger block size can yield less number of total blocks, which leads to less number of ciphertext blocks to decrease the ciphertext size. However, as a trade-off, it increases the decryption time due to the increased computation of discrete log search. Since we fix the block size, the decryption time is strictly linear to the message size which determines the number of message blocks.

As an encrypt-with-prove construction, SAVER.Enc is responsible for both *encryption* (Enc (sep)) and *proof* (Π_{snark}.Prove for SHA-256 of M). The zk-SNARK proving time Enc (sep) takes 0.74s, which is dominant in the total encryption time, while the separated encryption part Enc (sep) takes less than 8ms for $|M| = 2048bits$. Note that existing non-modular approaches require proving the encryption itself apart from the relation, which makes Enc (sep) much heavier; for instance, RSA-2048 using libsnark takes about $2.57s$ for Enc (sep).

In SAVER, the number of elements for PK, VK and $\mathcal{CT}$ is determined by the number of message blocks. Therefore it is shown in the result that $PK, VK, \mathcal{CT}$ size increases along with the message size. For the fixed relation, the size of common reference string (CRS) remains as 16MB for all message sizes, which is practical to be stored in the portable devices.

In general, the efficiency (e.g. encryption time, CRS size) of SAVER is much more practical compared to the typical approach. While the typical approach

requires a proving overhead which may take a few seconds and a few hundreds of MB, SAVER can encrypt the message within few milliseconds and few MB by avoiding the proving process.

6 Conclusion

This paper proposes SNARK-compatible verifiable encryption, which is a generic verifiable encryption achieved from modular combination of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) and encryption. The proposed SAVER provides modular composition with other proof systems by supporting commit-and-prove connection from LegoSNARK. SAVER also satisfies many useful functionalities. It is additively-homomorphic, so that the ciphertexts can be aggregated in an additive manner. It provides verifiable decryption, which can prove the validity of decryption to the public. It provides rerandomization, where the ciphertext can be rerandomized as a new encryption. The security of the proposed SAVER is formally proved under d-PKE and Poly assumptions.

Acknowledgements. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00518, Blockchain Privacy preserving techniques based on data encryption). J. Kim and H. Oh are co-corresponding authors.

References

1. G. Ateniese. Verifiable encryption of digital signatures and applications. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):1–20, 2004.
2. S. Bowe and A. Gabizon. Making groth's zk-snark simulation extractable in the random oracle model. Cryptology ePrint Archive, Report 2018/187, 2018. https://eprint.iacr.org/2018/187.
3. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Annual International Cryptology Conference*, pages 126–144. Springer, 2003.
4. M. Campanelli, D. Fiore, and A. Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. Cryptology ePrint Archive, Report 2019/142, 2019. https://eprint.iacr.org/2019/142.
5. A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020.
6. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, 1997.
7. A. Gabizon. On the efficiency of pairing-based proofs under the d-pke. Cryptology ePrint Archive, Report 2019/148, 2019. https://eprint.iacr.org/2019/148.
8. A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.
9. C. Gentry, S. Halevi, and V. Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. Cryptology ePrint Archive, Paper 2021/1397, 2021. https://eprint.iacr.org/2021/1397.
10. J. Groth. On the size of Pairing-Based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.
11. J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339, 2021. https://eprint.iacr.org/2021/339.
12. J. Groth and M. Maller. Snarky signatures: Minimal signatures of knowledge from Simulation-Extractable SNARKs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.
13. A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan. Non-interactive vss using class groups and application to dkg. Cryptology ePrint Archive, Paper 2023/451, 2023. https://eprint.iacr.org/2023/451.
14. J. Kim, J. Lee, and H. Oh. Simulation-extractable zk-snark with a single verification. *IEEE Access*, 2020.
15. A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. PAPAMAN-THOU, R. Pass, S. ABHI, and E. SHI. c: A framework for building composable zero-knowledge proofs. Technical report, Cryptology ePrint Archive, Report 2015/1093, 2015. http://eprint. iacr. org . . . , 2015.

16. A. E. Kosba, Z. Zhao, A. Miller, Y. Qian, T.-H. H. Chan, C. Papamanthou, R. Pass, A. Shelat, and E. Shi. How to use snarks in universally composable protocols. *IACR Cryptology ePrint Archive*, 2015:1093, 2015.
17. H. Lipmaa. Simulation-extractable snarks revisited. 2019.
18. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2111–2128, New York, NY, USA, 2019. Association for Computing Machinery.
19. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.
20. M. Prabhakaran and M. Rosulek. Rerandomizable rcca encryption. In *Annual International Cryptology Conference*, pages 517–534. Springer, 2007.
21. SCIPR-Lab. libsnark. https://github.com/scipr-lab/libsnark, 2014.

Appendix

A Definitions

A.1 Zero-Knowledge Succinct Non-interactive Arguments of Knowledge

For the zk-SNARK, we adopt the definitions from [10, 12].

Definition 3. *A zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) for $\mathcal{R}$ is a set of four algorithms $\Pi_{\mathsf{snark}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{SimProve})$ working as follows:*

- $(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$*: takes a relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$ as input and returns a common reference string CRS and a simulation trapdoor τ.*
- $\pi \leftarrow \mathsf{Prove}(CRS, \Phi, w)$*: takes a common reference string CRS, a relation $\mathcal{R}$, a statement and witness in the relation $(\Phi, w) \in \mathcal{R}$ as inputs, and returns a proof π.*
- $0/1 \leftarrow \mathsf{Vfy}(CRS, \Phi, \pi)$*: takes a common reference string CRS, a statement Φ, a proof π as inputs and returns 0 (reject) or 1 (accept).*
- $\pi \leftarrow \mathsf{SimProve}(CRS, \tau, \Phi)$*: takes a common reference string CRS, a simulation trapdoor τ, a statement Φ as inputs and returns a proof π.*

It satisfies completeness, knowledge soundness, zero-knowledge, and succinctness described as below:

Completeness: Given a true statement, a prover with a witness can convince the verifier. For all $\lambda \in \mathbb{N}$, for all $\mathcal{R}$ and for all $(\Phi, w) \in \mathcal{R}$, $Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), \pi \leftarrow \mathsf{Prove}(CRS, \Phi, w) : \mathsf{Vfy}(CRS, \Phi, \pi) = 1] = 1$.

Computational Knowledge Soundness: Computational knowledge soundness says that the prover must know a witness and such knowledge can be efficiently extracted from the prover by a knowledge extractor. Proof of knowledge requires that for every adversarial prover $\mathcal{A}$ generating an accepting proof, there must be an extractor $\chi_{\mathcal{A}}$ that, given the same input of $\mathcal{A}$, outputs a valid witness. Formally, an argument system Π_{snark} is computationally considered as knowledge sound if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\chi_{\mathcal{A}}$, such that $\mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}}, \mathcal{A}, \chi_{\mathcal{A}}}(\lambda)$ is negligible.

$$\mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}}, \mathcal{A}, \chi_{\mathcal{A}}}(\lambda) = Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), (\Phi^*, \pi^*) \leftarrow \mathcal{A}(CRS), w \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) : Vfy(CRS, \Phi^*, \pi^*) = 1 \wedge (\Phi^*, w) \notin \mathcal{R}] = negl(\lambda).$$

Perfect Zero-Knowledge: Perfect zero-knowledge states that the system does not leak any information besides the truth of the statement. This is modelled by a simulator that does not know the witness but has some trapdoor information that enables it to simulate proofs.

Succinctness: Succinctness states that the argument generates the proof of polynomial size in the security parameter, and the verifier's computation time is polynomial in the security parameter and in statement size.

A.2 Commit-Carrying SNARK and Commit-and-Prove SNARK

The commit-carrying SNARK (cc-SNARK) and the commit-and-prove SNARK (CP-SNARK) are originally defined in LegoSNARK [4]. The main concept of the LegoSNARK is to connect the modular SNARK systems and ensure the same input between them via commitments.

The cc-SNARK is a proof system where the proof includes a commitment to the portion of witnesses, which can be used for the connection. There is no specific restriction for the commitment scheme, but it is convenient to consider it as the Pedersen vector commitment $c = h^o \cdot h_1^{u_1} \cdots h_n^{u_n}$ for a random $o \xleftarrow{\$} \mathbb{Z}_p^*$, messages $u_1, \cdots, u_n$ and generators $h, h_1, \cdots, h_n$.

Definition 4. *(cc-SNARK: Definition 3.2 of [4]). A commit-carrying zk-SNARKs for a relation $\mathcal{R}$ is a set of four of algorithms $\Pi_{\mathsf{cc}} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{VerProof}, \mathsf{VerCommit})$ that works as follows:*

- $(ck, ek, vk) \leftarrow \mathsf{KeyGen}(\mathcal{R})$: *takes a relation $\mathcal{R}$ as inputs, and outputs a common reference string which includes a commitment key ck, an evaluation key ek, and a verification key vk.*
- $(c, \pi; o) \leftarrow \mathsf{Prove}(ek, x, w)$: *takes an evaluation key ek, a statement x and a witness $w := (m, \omega)$ such that the relation $\mathcal{R}$ holds as inputs, and outputs a proof π, a commitment c and an opening o such that $\mathsf{VerCommit}(ck, c, u, o) = 1$.*
- $0/1 \leftarrow \mathsf{VerProof}(vk, x, c, \pi)$: *takes a verification key vk, a statement x, a commitment c, a proof π as inputs, and outputs 1 if x, c, π is within the relation $\mathcal{R}$, or 0 otherwise.*
- $0/1 \leftarrow \mathsf{VerCommit}(ck, c, u, o)$: *takes a commitment key ck, a commitment c, a message u and an opening o as inputs, and outputs 1 if the commitment opening is correct, or 0 otherwise.*

which satisfies completeness, succinctness, knowledge soundness, zero knowledge and binding (described in [4]).

The CP-SNARK is a proof system that can link existing cc-SNARKs by using their commitments. The LegoSNARK defines the CP-SNARK framework and provides a CP-SNARK scheme CP, that guarantees the connectivity between multiple cc-SNARKs via Pedersen vector commitments.

Definition 5. *(CP-SNARK: Definition 3.1 of [4]). Let $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of relations R over $\mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_\omega$ such that $\mathcal{D}_u$ splits over l arbitrary domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_l)$ for some arity parameter $l \geq 1$. Let $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ be a commitment scheme (as per Definition 2.1) whose input space $\mathcal{D}$ is such that $\mathcal{D}_i \subset \mathcal{D}$ for all $i \in [l]$. A commit and prove zk-SNARK for Com and $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ is a zk-SNARK for a family of relations $\{\mathcal{R}_\lambda^{\mathsf{Com}}\}_{\lambda \in \mathbb{N}}$ such that:*

- *every $\boldsymbol{R} \in \mathcal{R}^{\mathsf{Com}}$ is represented by a pair (ck, R) where $ck \in \mathsf{Setup}(1^\lambda)$ and $R \in \mathcal{R}_\lambda$;*

– $\boldsymbol{R}$ *is over pairs* $(\boldsymbol{x}, \boldsymbol{w})$ *where the statement is* $\boldsymbol{x} := (x, (c_j)_{j\in[l]}) \in \mathcal{D}_x \times \mathcal{C}^l$*, the witness is* $\boldsymbol{w} := ((u_j)_{j\in[l]}, (o_j)_{j\in[l]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_l \times \mathcal{O}^l \times \mathcal{D}_\omega$*, and the relation* $\boldsymbol{R}$ *holds iff*

$$\bigwedge\nolimits_{j\in[l]} \mathsf{VerCommit}(ck, c_j, u_j, o_j) = 1 \wedge \mathcal{R}(x, (u_j)_{j\in[l]}, \omega) = 1$$

We denote a CP-SNARK as a triple of algorithms $\mathsf{CP} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{VerProof})$ *as follows.*

– $(ek, vk) \leftarrow \mathsf{KeyGen}(ck, R)$: *takes a commitment key* ck*, a relation* $\mathcal{R}$ *as inputs, and outputs a common reference string which includes an evaluation key* ek*, and a verification key* vk*.*
– $\pi \leftarrow \mathsf{Prove}(ek, x, (c_j)_{j\in[l]}, (u_j)_{j\in[l]}, (o_j)_{j\in[l]}, \omega)$: *takes an evaluation key* ek*, a statement* x*, commitments* c_j*, messages* u_j*, randoms* o_j*, witnesses* ω*, and outputs the proof of correct commitment.*
– $0/1 \leftarrow \mathsf{VerProof}(vk, x, (c_j)_{j\in[l]}, \pi)$: *takes a verification key* vk*, a statement* x*, commitments* c_j*, a proof* π*, and rejects or accepts the proof.*

A.3 Commit-Carrying Encryption

We define a new notion of encryption that can output a commitment which shares a same format with the commit-carrying SNARKs. If the encryption scheme is capable of outputting the Pedersen vector commitment from existing ciphertexts, we say that it is a *commit-carrying encryption.* It can be formally defined as follows:

Definition 6. *Suppose a public-key encryption scheme* Π_{enc} *which outputs a ciphertext* $\mathcal{CT}$ *for the message input* m*. For the ciphertext* $\mathcal{CT}$ *and a Pedersen vector commitment* $c = \mathsf{Ped.Commit}(m)$ *of the message* m*, if there exists an efficient polynomial-time function* $f_c(x)$ *which satisfies* $f_c(\mathcal{CT}) = c$*, we say that the encryption scheme* Π_{enc} *is a commit-carrying encryption.*

The commit-carrying encryption follows the definition of standard public-key encryption, but it also gains modular composability between other commit-carrying systems via commit-and-prove SNARK (CP-SNARK).

A.4 Verifiable Decryption

We refine the definition of verifiable decryption from [3]; the definition in [3] represents the proof system and the encryption system separately, but we intend to combine them as an encryption scheme with verifying phase. Plus, we strengthen the security notion from decryption soundness to perfect decryption soundness, and introduce a new security notion - perfect zero-knowledge.

Definition 7. *Suppose we have an encryption scheme* Π_{enc} *which satisfies the definition of standard public-key encryption. We say that* Π_{enc} *is a verifiable decryption* Π_{VD}*, if it additionally includes the following polynomial-time algorithm:*

- $M, \nu \leftarrow \mathsf{Dec}(SK, \mathcal{CT})$: *the decryption of a ciphertext* $\mathcal{CT}$ *outputs a message* M*, along with the corresponding decryption proof* ν*.*
- $0/1 \leftarrow \mathsf{Verify\_Dec}(VK, M, \nu, \mathcal{CT})$: *takes a verification key* VK*, a message* M*, a decryption proof* ν*, a ciphertext* $\mathcal{CT}$ *as inputs, and outputs 1 if* M, ν *is a valid decryption for* $\mathcal{CT}$ *or 0 otherwise.*

with satisfying completeness, and perfect decryption soundness, and indistinguishability as described below:

Completeness: A message M and a decryption proof ν must pass the verification, if decrypting $\mathcal{CT}$ with SK outputs M, formally as $Pr[(M, \nu) \leftarrow \mathsf{Dec}(SK, \mathcal{CT}), \mathcal{CT} = \mathsf{Enc}(PK, M) : \mathsf{Verify\_Dec}(VK, M, \nu, \mathcal{CT}) = 1] = 1$.

Perfect Decryption Soundness: The advantage of an adversary forging verifying $M^*, \nu^*, \mathcal{CT}^*$ where M^* is not a decryption of $\mathcal{CT}$ is 0.

$$\mathbf{Adv}^{sound}_{\Pi_{\mathsf{VD}},\mathcal{A}}(\lambda) = Pr[(M^*, \nu^*, \mathcal{CT}^*) \leftarrow \mathcal{A}(SK, PK, VK) : \mathsf{Verify\_Dec}(VK, M^*, \nu^*, \mathcal{CT}^*) = 1 \wedge \mathsf{Dec}(SK, \mathcal{CT}^*) \neq M^*] = 0.$$

Indistinguishability: A verifiable decryption should satisfy IND-CPA of the original public-key encryption, with providing additional information ν to an adversary $\mathcal{A}$, for $\mathcal{A}$'s chosen messages.

A.5 Rerandomizable Encryption

We adopt the definition of rerandomizable encryption from [20].

Definition 8. *Suppose we have an encryption scheme* Π_{enc} *which satisfies the definition of standard public-key encryption. We say that* Π_{enc} *is a rerandomizable encryption* Π_{RR}*, if it additionally includes the following polynomial-time algorithm:*

- $\mathcal{CT}' \leftarrow \mathsf{Rerandomize}(PK, \mathcal{CT})$: *a randomized algorithm which takes a public key* PK *and a ciphertext* $\mathcal{CT}$ *and outputs another ciphertext* $\mathcal{CT}'$*.*

which satisfies completeness and rerandomizability described as below:

Completeness: For every ciphertext $\mathcal{CT}$ and every $\mathcal{CT}'$ in the support of $\mathsf{Rerandomize}(PK, \mathcal{CT})$, we must have $\mathsf{Dec}(SK, \mathcal{CT}') = \mathsf{Dec}(SK, \mathcal{CT})$.

Rerandomizability: For every plaintext M and every ciphertext $\mathcal{CT}$ in the support of $\mathsf{Enc}(PK, M)$, the distribution of $\mathsf{Rerandomize}(PK, \mathcal{CT})$ is identical to another round of $\mathsf{Enc}(PK, M)$.